A Model for Dynamic Hyperspaces

Ruzanna Chitchyan, Ian Sommerville, Awais Rashid
Computing Department, Lancaster University, Lancaster, LA1 4YR, UK
{rouza | is | awais}@comp.lancs.ac.uk

Abstract

This paper proposes a composition mechanism for hyperslices decomposed in accordance with the Hyperspaces approach. Our composition mechanism aims to maintain each concern (including composition concerns) as a first class entity all through its lifetime, allowing for dynamic change and re-configureability of the resultant systems. In developing this model we draw on the concern decomposition mechanism of the Hyperspaces approach, message interception and manipulation ideas of the Composition Filters approach and component integration mechanism of Connectors.

1. Introduction

Hyperspaces [14] provide an elegant technique of decomposing systems into a set of single-minded modules each of which is an independent concern. In the manual for the current implementation of Hyperspaces approach – HyperJ – it is stated that "The multi-dimensional approach permits integration and customisation using any concerns, in any dimensions…. This ability to treat all concerns as equal provides developers the ability to focus their attention on precisely the part of a system they care about..."

However, we have observed that so far most of the work on Multi-dimensional Separation of Concerns has focused on user domain concerns, i.e. those concerns that are obtained from the requirements provided by the system users. What has been overlooked is that system development also involves developer-specific concerns. Although (usually) invisible to the system end-users, these concerns are vital for system developers' work. Consequently, we believe that developer-specific concerns should be incorporated into the working set of concerns for any software system. Concern composition is one such example: since it is itself a well-defined developer-related concern, it needs to be addressed explicitly and treated "as equal" with all other concerns.

The current implementation of HyperJ does not explicitly account for developer-specific concerns. Composition concerns, for instance, are not clearly identifiable entities in the hyperspace of concerns, but are transitory (hyperspace definition, concern mapping and hypermodule composition) files whose content is mapped into changes introduced to the class files of user-defined concerns. Hence, this method forces user-defined concern re-definition in the resultant composed system in order to accommodate the composition requirements of the developers.

This composition technique has some shortcomings, as it infringes:

- run-time traceability of primary concerns – in the composed system the primary requirements are distorted by changes due to the composition process
- reuse and understandability of the composite concerns - if due to requirements evolution, a particular primary requirement is amended, the whole composed system (and its composition) could be affected.
- runtime reconfigurability of the system – it is impossible to add/remove new concerns without re-composition of the system with new composition requirements mapped to the corresponding concerns' class files.

We are of the view that the above problems can be resolved by treating all types of concerns (both user and developer-defined) as first class entities. Specifically, in this paper we propose a new model of Hyperspace-type concern composition. We intend to maintain autonomy of each concern all through its lifetime, encapsulating the composition-related issues into separate first class entities. This decouples concerns making it possible to accommodate any composition-related change through composition connector adaptation.

In developing this model we build on our earlier work on utilising strengths of multiple separation of concerns' mechanisms [4] [16]. We draw on the concern decomposition mechanism of the Hyperspaces approach [14], message interception and manipulation ideas of the Composition Filers approach [2] and component integration mechanism of Connectors [5], all of which we believe are pertinent to achieving clean hyperslice composability and promoting reusability of coarse-grained (i.e. composed) concerns.

Although this model could possibly be used to resolve many other developer-related concerns (e.g. deployment, etc.), the discussion in the present paper is limited to the composition concerns.

The rest of this paper is structured as follows: in section 2 we briefly present the work which our model draws upon, section 3 describes the model itself, the related work is briefly discussed in section 4, and section 5 presents future work and conclusions.

2. Background

In developing the Dynamic Hyperspaces model a number of technologies have been used. This section provides a brief description of these technologies.

2.1 Hyperspaces

This approach proposes to use a set of modules each of which address a single concern (called hyperslice) to encapsulate concerns in dimensions other than the one used for the dominant formalism [11] [14]. The modules within a hyperslice are standard modules written in the chosen formalism, but these modules contain only units pertaining to the concern addressed by the hyperslice. Hyperslices can overlap, i.e. a given unit in a hyperslice can appear, possibly in a different form, in other hyperslices. Although hyperslices do not always fulfil the completeness requirements of the chosen formalism, they observe "declarative completeness" – declaring (but not necessarily implementing) everything that is used within the hyperslice. All the concerns of importance are modelled as hyperslices, which are then composed to produce a complete system. At the composition stage issues such as overlapping are resolved via composition rules.

Although this model can be instantiated in any formalism, to the best of our knowledge, it has only been done for the OO paradigm. The composition tool, called HyperJ [15], for this instantiation has also been developed.

In our model we maintain the decomposition principles of Hyperspaces. We also clearly define two types of concerns: user and developer-related concerns. Both types need to be treated as first class entities all through the software development and maintenance process. We differ in our composition approach (see section 3) from that implemented by HyperJ. This is because the composition mechanism provided by HyperJ is static and requires manual intervention for change introduction. The developer has to specify the corresponding elements and the composition mechanism for each of the composeable elements (more generic strategy specification such as "merge by name" can also be used). As mentioned earlier, the composition-related concerns are not treated as equal members of concerns' hyperspace. Although hyper-module composition is specified in a separate composition file, it is only a transitory unit. Consequently, when the elementary concerns are composed, they get contaminated with properties of the composite concern. This is illustrated in Figure 1 below:

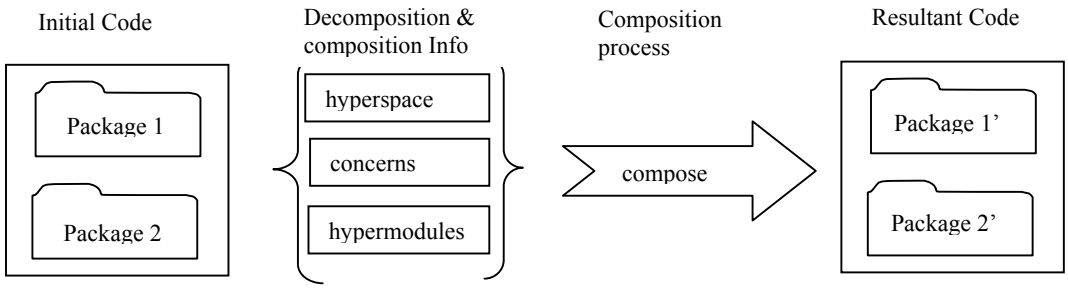


Figure 1: packages 1 and 2 are composed in accordance with information provided in the composition section. Through the composition process the packages are transformed into Package 1' and Package 2' with their class code content amended and composition section information dissolved in them.

2.2 Composition Filters

The Composition Filters (CF) model [2] extends the Object-Oriented model in a modular and orthogonal way. Since behaviour in the OO model is implemented by exchanging messages between objects, the CF model proposes to use a set of input and output filters for message interception and manipulation. By wrapping these filters around the objects, CFs are able to manipulate object behaviour without directly invading object implementation.

Composition Filters are very effective in implementing concerns involving message interception and execution of actions before and after executing a method. Their key characteristic is the ability to operate at instance granularity and attachment on a per-instance basis. A number of filter types are provided, for instance, dispatch filters are used for delegating messages, wait filters for buffering messages etc.

In our model we utilise the message interception and manipulation capabilities of Composition Filters. A filter can represent a particular primary concern in the composition concerns dimension, for instance, a concern to intercept and dispose of a given type of message. A composed set of filters can serve as a compound composition concern. Thus, filters are units in the Composition concerns dimension which can

be contained in our composition connectors (see section 3). Unlike the CF approach, we do not *require* an object to attach a filer to. Our composition concerns interface hyperslices, which could be objects in some cases, but could also be any other type of concern. Also, in this model user-defined filter types are allowed.

2.3 Connectors

The concept of connectors [1] originates from the area of software architecture. Initially connectors were proposed to facilitate component integration by catering for specific features of interactions among components in a system. Further on, in [13] it was suggested to provide connectors with first class entity status because they contribute towards the better understandability of system architecture through [13]:

- Localising information about interactions of components in a system (which otherwise is spread across interacting components), this in turn facilitates change and maintenance;
- Capturing the design decisions and rules of interactions amongst components, which facilitates analysis and maintenance;
- Handling incompatibilities between components;
- Providing natural support for multi-language and multi-paradigm systems;
- Supporting dynamic changes in system connectivity;
- Supporting mobility (only connectors know the network location of individual components) and increasing component reusability (by separating communication apparatus from components) [10];
- Solving deployment anomaly problems [5].

Connectors are grouped [8] into: implicit [7], set of built-in connectors [12], and user-defined [10].

The composition concerns in our model are connectors for hyperslices. We use the concept of a connector - a gluing component between independent entities - to combine separate hyperslices into a hypermodule. A connector consists of elements (e.g. sets of filters, composition strategies, etc.) from the Composition Concerns space; it should be able to reflect upon its connected hyperslices.

It is worth noting that our connectors can yield all the above listed connector-related advantages to our model.

3. Dynamic Hyperspaces Model

Our intent is to provide a composition mechanism that will allow all the primary concerns (i.e. concerns that cannot be further decomposed to other meaningful concerns) to endure in the composite concerns after composition. We propose to decouple individual concerns through explicitly separated composition encapsulation units.

We follow the Hyperspaces decomposition approach in separating concerns into single-minded hyperslices (or primary concerns). We also require that an additional dimension for Composition concerns is specified in each Hyperspace-type decomposition. This additional dimension contains connector-concerns. At the composition stage the connector concerns are used to compose other (primary or composite) concerns. Connectors can be composed into composite connectors. In order to facilitate new connector definition, a number of frequently used composite connectors and connector integration strategies will be provided.

It is expected that an item from the Composition concerns' space will be utilised to perform each composition step in the system integration strategy. Similar to the Composition Filters object interaction strategy, neither preliminary nor composite concerns will be able to interact with other user-domain concerns directly; any type of interaction will be channelled through a set of connectors[1]. Thus, a connector will be able to reflect upon primary hyperslices immediately connected to it. While, at the same time, the hyperslice internals will be hidden from all other connectors and hyperslices.

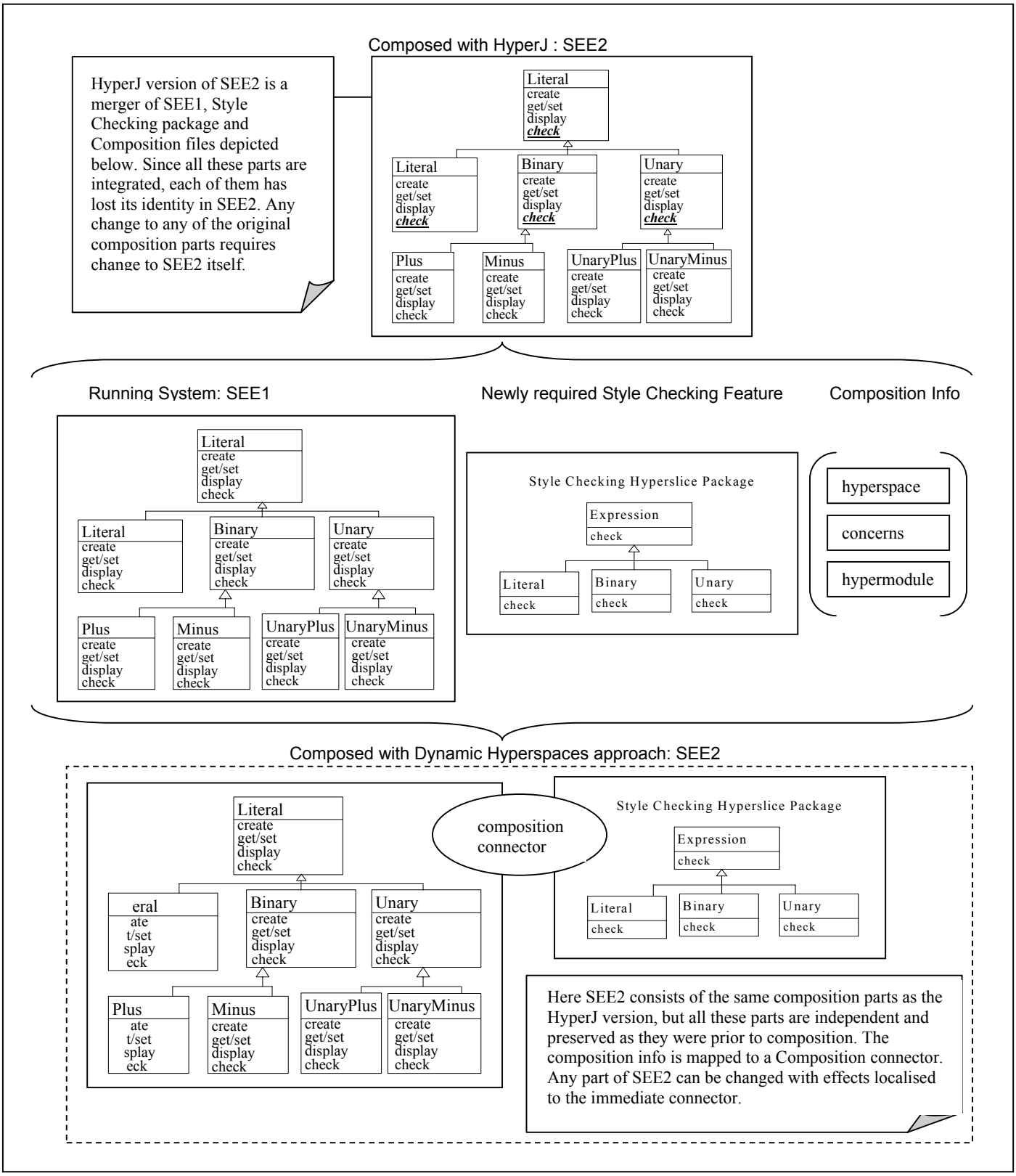
Briefly stated, the goal of the Dynamic Hyperspaces model is to provide a composition mechanism where:

- composition concerns (along with other types of user/developer concerns) are first class entities;
- hyper-slice integration information is retained in independent composition concerns - connectors;
- all concerns used in the system are retained unchanged during runtime;
- interactions between the concerns are resolved through their connectors;
- the model is open and extensible.

[1] It should be noted that in this model the multiplicity of compositions has not been decisively resolved yet. The final solution to this issue will depend on structure of connectors and integration strategies which are still to be developed.

In order to illustrate the use of the proposed model we demonstrate an example and contrast it to the HyperJ implementation (Figure 2). To allow for a common reference we use the Software Engineering Environment (SEE) example, detailed in the HyperJ manual [15]. We assume that we are at the stage of adding Style Checking [15, section 5.3] feature to an already running system. Due to space constraints we will not discuss how the running system itself is composed.

Figure 2: An example of Dynamic Hyperspaces vs. HyperJ composition:



To meet our composition requirements, the connectors need to be able to accommodate:

- dynamic introduction/removal of hyperslices into/from a composite hyperslices, limiting need for any change propagation. At a coarser granularity level, any new component that is aligned with the underlying model, can be integrated into/removed from the system through a connector.
- new connector type and composition mechanism creation. In order for the model to be open and extensible, user defined connector types should be allowed. This implies that new composite connectors should be composeable from basic pre-defined types provided in the model, allowing developers to construct connectors suited for their needs.

Our connectors decouple hyperslices in a hypermodule and promote reuse of individual concerns and their compositions. In fact each hyperslice in our model can be viewed as a fine-grained component. These fine-grained components can be composed into progressively larger components through connectors.

This model can reduce maintenance and evolution problem of component-based development. Since at any level each of the primary concerns maintains its identity unchanged, changing requirements can be easily mapped onto a composite component by removing the out-of-date primary concern and replacing it with an updated one. In the process of primary component replacement only its immediate connectors will be affected. These connectors should be able to absorb the necessary changes. Similarly a composite part of a composite component can be replaced with change localised in its immediate connectors.

It should be noted that the drawback of our (as well as reflection and wrapper-based techniques) is the possible significant performance overhead caused by the large number of indirect referencing. Some ways of alleviating this problem include mapping most frequently used Composition concerns onto optimised language constructs and optimisation of composition strategies. If performance is of prime importance, techniques such as the code injection should be used instead.

4. Related work

At present composition filters are best suited for dynamic evolution. The higher-level filtering patterns (e.g. dispatch filters, error filters, etc.) draw a clear picture of the results of object composition. The dynamic evolution is supported through a variation of interception filters

Composition Filters compose objects into systems with desired behaviour through manipulation of object message exchange. Our model composes hyperslices (i.e. concerns, which could or could not be representing objects) into new hyperslices (i.e. composite concerns, possibly objects) and their joint behaviour into a single new concern.

Another approach supporting dynamic evolution is JAC (Java Aspect Components) which uses wrappers and reflection to achieve the dynamism. However JAC defines components at much coarser level then we do. They define components as only entire crosscutting concerns, such as synchronisation and persistence, while our definition includes a much wider set of items, from fine to coarse grain.

Other work on dynamic aspects includes Orleans and Lieberherr's report on work on DJFramework: an extension to adaptive programming offering dynamic traversal strategies [9]. Besides, work is currently being carried out to integrate dynamic aspects in AspectJ. Neither of these approaches explicitly identified composition concerns as first class entities.

Some dynamic byte code adaptation approaches (e.g. JMangler [6] and Javassist [3]) have also been developed.

5. Conclusion.

In this paper we distinguish user- and developer-related types of concerns. We propose to treat them "as equal". However, it is evident from our model that there are differences between these types: developer-defined concerns are meaningful only in reference with user-defined concerns (e.g.: there is no point of composition concerns if there are not other concerns to be composed together). It is also likely that many of developer-specific concerns arise due to software development technology used by the developers.

Nevertheless, we also suggest that giving all concerns a first class entity status presents our model a number of advantageous Software Engineering properties, allowing it to:

- retain independence of hyperslices all through the software development lifecycle, thus promoting traceability of concerns all through their life cycle;
- provide for dynamic reconfiguration of compositions of hyperslices /hyperspace-based systems, without intervention into the individual hyperslices, by simply changing the hyperspace composition information retained in the corresponding connector, thus providing for good evolvability of the system;

- provide user-defined composition strategies through user-defined connector types, improving flexibility;
- provide a uniform composition mechanism for both fine and coarse grained entities: from integration of individual concerns to large components;
- promote reuse of desired type of concerns.

Our model distinguishes composition neutral and composition dependant parts, where all primary concerns are of the first type and only primary and composite connectors are composition dependant. This suggests a good general guideline that user-defined concerns should be independent of developer-defined concerns.

Our on-going work is focused on developing a meta-model for the Dynamic Hyperspaces model presented above and refining the composition mechanism.

As part of our future work we intend to implement a system that realises the developed model, construct a mechanism that will allow to incorporate domain specific knowledge into the composition process, and apply this model to other developer-related concerns.

References:

[1] Allen R. J.: "A Formal approach to Software Architecture" Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1997.

[2] L. Bergmans and M. Aksit, "Composing Crosscutting Concerns using Composition Filters", Communications of the ACM, Vol. 44, No. 10, pp. 51-57, 2001.

[3] S. Chiba, "Load-time Structural Reflection in Java", 14th European Conference on Object-Oriented Programming (ECOOP'00), 2000, Springer-Verlag, Lecture Notes in Computer Science, 1850, pp. 313-336.

[4] R. Chitchyan, I. Sommerville, and A. Rashid, "An Analysis of Design Approaches for Crosscutting Concerns", Workshop on Aspect-Oriented Design (held in conjunction with the 1st Aspect Oriented Software Development Conference AOSD 2002), 2002.

[5] D. Balek: "Connectors in Software Architectures" PhD thesis, March 2002.

[6] G. Kniesel, P. Contanza, and M. Austermann, "JMangler - A Framework for Load-Time Transformation of Java Class Files", IEEE Workshop on Source Code Analysis and Manipulation (SCAM), 2001.

[7] Magee J., Dulay N., Kramer J, : Regis: A constructive Development Environment of Distributed Programmes, In Distributed Systems Engineering Journal , 1 (5), 1994.

[8] Medvidovic N., Taylor R. N.: "A Framework for Classifying and comparing Architecture Description Languages" In Proceeding s of the 6th European Software Engineering Conference/5th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Zurich, Switzerland, 1997.

[9] D. Orleans and K. J. Lieberherr, "DJ: Dynamic Adaptive Programming in Java", 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns (Reflection 2001), 2001, Springer-Verlag, Lecture Notes in Computer Science, 2192, pp. 73-80.

[10] Oreizy, P., Rosenblum, D. S., Taylor, R.N. : On the Role of Connectors in Modelling and Implementing Software Architectures, Technical Report UCI-ICS-98-04, University of California, Irvine, 1998.

[11] H. Ossher and P. Tarr "Multi-Dimensional Separation of Concerns using Hyperspaces". IBM Research Report 21452, April, 1999.

[12] Shaw M., DeLine R., Klein D. V., Ross T. L, Young D. M., Salesnki G.: "Abstractions for Software Architectures and Tools to Support Them ". IEEE Transaction of Software Engineering, Vol. 21, No. 4., April 1995, pp 314-335.

[13] Shaw, M: Procedure Calls are they Assembly Language of Software Interconnections: Connectors Deserve First-Class Status. In D.A. Lamb (ed) Studies of Software Design, proceedings of a 1993 Workshop, Lecture Notes in Computer Science no 1078, Springer-Verlag 1996.

[14] P. Tarr, H. Ossher, W. Harrison and S.M. Sutton, Jr. "N Degrees of Separation: Multi-Dimensional Separation of Concerns". Proceedings of the International Conference on Software Engineering (ICSE'99), May, 1999.

[15] P. Tarr & H. Ossher "Hyper/J User and Installation Manual" http://www.research.ibm.con/hyperspace

[16] A. Rashid, "A Hybrid Approach to Separation of Concerns: The Story of SADES", 3rd International Conference on Meta-Level Architectures and Separation of Concerns (Reflection), 2001, Springer-Verlag, Lecture Notes in Computer Science, 2192, pp. 231-249.